TP1 : Algorithme de Quine

Objectifs du TP :

- Implémenter l'algorithme de Quine et étudier sa terminaison, sa correction et sa complexité.
- Faire le lien entre un problème de graphes et un problème de satisfiabilité.
- Réviser l'utilisation de types sommes en Ocaml.

Un problème important en logique propositionnelle est la question de savoir si une formule F donnée est satisfiable : c'est le problème SAT. Lors d'un TP, nous avions élaboré un algorithme naïf de test de satisfiabilité d'une formule qui consistait à énumérer toutes les valuations v possibles pour F et à vérifier pour chacune si $v(F) = 1$. Cet algorithme a une complexité au moins exponentielle en le nombre n de variables de F puisqu'il nous faut tester 2^n valuations dans le pire cas avant de pouvoir conclure.

Nous avons également décrit l'année dernière un autre algorithme permettant de tester la satifiabilité d'une formule : l'algorithme de Quine. Nous n'avions toutefois pas étudié cet algorithme en détail, par faute d'outils adéquats à l'époque. Ce TP est donc un complément au cours sur l'algorithme de Quine.

Les parties 3 et 4 sont à rendre au format papier pour le 23/09 en début de TP. Les questions 14 et 21 sont facultatives.

Partie 1 *Mise en jambe : manipulation de formules*

On choisit dans ce sujet de représenter une formule du calcul propositionnel à l'aide du type Ocaml suivant. On se restreint à des formules n'utilisant pas le connecteur $\Leftrightarrow$ ce qui n'est pas bien grave puisque $\{\neg, \vee, \wedge, \Rightarrow\}$ est complet.

```
type formule =
  |C of bool
  |V of int
  |Et of formule * formule
  |Ou of formule * formule
  |Imp of formule * formule
  |Non of formule
```

1. Déclarer une variable f_ex représentant la formule F_{ex} suivante :

$$(v_0 \Rightarrow (v_1 \wedge (\neg v_0 \vee v_2))) \wedge \neg(v_0 \wedge v_1)$$

2. Ecrire une fonction taille de signature formule -> int telle que taille f renvoie la taille de la formule f (qui, on le rappelle, correspond à la taille de l'arbre syntaxique associé à cette formule).
3. Ecrire une fonction var_min de signature formule -> int telle que var_min f renvoie l'indice minimal parmi les indices des variables intervenant dans f. Si la formule ne contient aucune variable, on déclenchera une erreur explicite à l'aide de failwith par exemple.

Partie 2 *Algorithme de Quine : implémentation*

On rappelle que l'algorithme de Quine consiste à construire un arbre de décision binaire — dit "arbre de Quine de F" — à partir d'une formule du calcul propositionnel F de la manière suivante :

- On simplifie F en une formule $s(F) = G$ en utilisant récursivement les opérations syntaxiques ci-dessous à partir des feuilles de l'arbre syntaxique de F jusqu'à sa racine. Ces dernières visent à simplifier F jusqu'à ce qu'elle soit réduite à une constante ($\bot$ ou $\top$) ou ne contienne plus de constante.
- Si G est réduite à une constante, l'arbre de décision associé à F est réduit à une feuille étiquetée par vrai ou faux selon la valeur de la constante.
- Sinon, on choisit une variable v apparaissant dans G, d'indice i, et on calcule les arbres de décision $a_\bot$ et $a_\top$ associés à $G[\bot/v]$ et $G[\top/v]$ respectivement. L'arbre de décision associé à F est alors $(i, a_\bot, a_\top)$.

Remarque : On choisit ici de stocker dans un noeud de l'arbre de décision l'indice i de la variable substituée à l'étape correspondante de l'algorithme mais on peut aussi y stocker la formule dans laquelle on est en train de substituer la variable d'indice i. Les simplifications syntaxiques permettant de passer de F à $s(F)$ sont ici rappelées. La suite de symboles $A \to B$ signifie "on remplace la formule A par la formule B" :

$\varphi \wedge \bot \to \bot$	$\varphi \vee \top \to \top$	$\varphi \Rightarrow \top \to \top$	$\neg\top \to \bot$
$\bot \wedge \varphi \to \bot$	$\top \vee \varphi \to \top$	$\top \Rightarrow \varphi \to \varphi$	$\neg\bot \to \top$
$\varphi \wedge \top \to \varphi$	$\varphi \vee \bot \to \varphi$	$\bot \Rightarrow \varphi \to \top$	
$\top \wedge \varphi \to \varphi$	$\bot \vee \varphi \to \varphi$	$\varphi \Rightarrow \bot \to \neg\varphi$	

4. Dessiner l'arbre de décision obtenu pour la formule F_{ex} en choisissant systématiquement de substituer la variable d'indice minimal intervenant dans la formule considérée.
5. Rappeler (sans preuve) à quelle condition sur l'arbre obtenu pour F cette formule est satisfiable. Reprendre la question en remplaçant "est satisfiable" par "est une tautologie".
6. Ecrire une fonction elim_constantes de signature formule -> formule permettant de calculer $s(F)$ à partir d'une formule F.
7. Ecrire une fonction substitue de signature formule -> int -> formule telle que substitue f i g renvoie la formule f dans laquelle la variable i a été substituée par la formule g.

On représente un arbre de décision binaire à l'aide du type Ocaml suivant :

```
type arbre_decision =
  |Feuille of bool
  |Noeud of int * arbre_decision * arbre_decision
```

8. Ecrire une fonction arbre_Quine de signature formule -> arbre_decision prenant en entrée une formule F et renvoyant l'arbre de décision associé à F selon l'algorithme de Quine. On choisira à chaque étape de substituer la variable d'indice minimal présente dans la formule considérée.
9. En déduire une fonction satisfiable de signature formule -> bool indiquant si la formule prise en entrée est satisfiable ou non. Indiquer les changements qu'il suffirait d'effectuer pour obtenir plutôt une fonction indiquant si la formule prise en entrée est tautologique.

Partie 3 *Algorithme de Quine : analyses*

10. Prouver par induction que, si une formule F ne contient aucune variable, alors sa simplification $s(F)$ selon les règles de Quine est soit $\bot$, soit $\top$.
11. Justifier rigoureusement la terminaison de l'algorithme de Quine.
12. Prouver l'affirmation suivante : "le sous-arbre enraciné en la formule F admet au moins une feuille étiquetée par vrai (ou $\top$ si on considère qu'on étiquette les noeuds par des formules) si et seulement si F est satisfiable".
13. Donner une majoration pour la complexité pire cas de l'algorithme de Quine en fonction de la taille de la formule prise en entrée.
14. (facultatif) Exhiber en justifiant une formule F pour laquelle dans le pire cas (c'est-à-dire, si on substitue malencontreusement les variables dans un ordre peu propice) l'algorithme de Quine est exponentiel en la taille de F. Pourquoi est-ce attendu ?

Partie 4 *Algorithme de Quine : application*

Un problème sous contraintes peut fréquemment se réécrire sous la forme d'un problème de satisfiabilité d'une formule bien choisie. On s'intéresse ici au problème de coloriage de graphes non orientés. On les représentera en Ocaml par listes d'adjacence à l'aide du type suivant :

```
type graphe = int list array
```

Si $k \in \mathbb{N}$, on rappelle qu'un graphe est k-coloriable s'il est possible d'attribuer à chacun des sommets un numéro entier de $[\![0, k-1]\!]$ de telle sorte à ce que deux sommets voisins ne portent jamais le même numéro.

Si G est un graphe à n sommets et k un entier, on cherche à déterminer une formule $F_{G,k}$ qui soit satisfiable si et seulement si G est k-coloriable. Pour ce faire, on introduit pour tout $i \in [\![0, n-1]\!]$ et tout $c \in [\![0, k-1]\!]$ la variable $v_{i,c}$ exprimant le fait que le sommet i est colorié avec la couleur c. Pour tout sommet i, on définit la formule

$$A_i = \bigvee_{c=0}^{k-1} v_{i,c}$$

et pour tout sommet i et toute couleur c, on définit la formule

$$B_{i,c} = v_{i,c} \Rightarrow \neg \left(\bigvee_{c' \neq c} v_{i,c'} \vee \bigvee_{i \text{ et } j \text{ sont voisins}} v_{j,c} \right)$$

15. Que représentent sémantiquement les formules A_i et $B_{i,c}$?
16. A partir de ces formules, construire une formule $F_{G,k}$ satisfiable si et seulement si G est k coloriable.
17. Ecrire une fonction `encode` de signature `graphe -> int -> formule` prenant en entrée un graphe G et un entier k et renvoyant la formule $F_{G,k}$. On expliquera précisément son fonctionnement. On n'hésitera pas à utiliser les primitives Ocaml sur les listes, à décomposer la fonction `encode` en sous-fonctions et on pourra numéroter les variables $v_{i,c}$ à l'aide de la fonction :

    ```
    let variable i c k = V (i*k + c)
    ```

18. En déduire une fonction `est_k_coloriable` de signature `graphe -> int -> bool` prenant en entrée un graphe G et un entier k et indiquant si G est k coloriable.

On appelle *nombre chromatique* d'un graphe, le plus petit entier k tel qu'il soit k-coloriable. Le nombre chromatique d'un graphe est bien défini étant donné que tout graphe est coloriable à l'aide de $d+1$ couleurs où d est le degré maximal d'un de ses sommets.

19. Ecrire une fonction `nb_chromatique` de signature `graphe -> int` calculant le nombre chromatique du graphe G en entrée. Expliquer la démarche utilisée.

20. Donner le nombre chromatique du graphe `graphe_mystere` proposé dans le fichier `TP1_sujet.ml` mis à disposition dans l'espace partagé.

21. (facultatif) Proposer et implémenter une ou plusieurs stratégies destinées à améliorer autant que possible la complexité en pratique de la fonction `satisfiable`.